IJAAR PUBLISHING®
Scholarly Journal of Science and Technology Research & Development | ISSN: 2955-0807
Vol. 5, Issue 3 (2026) | https://doi.org/10.46654/ac0w3647

From Detection to Defense: A Comprehensive Benchmark and Study of Static Analysis Tools for PHP WebShell Detection

[1]**Orji, Cyrus Ebere**, MCPN
[1]*Department of Computer Science, Imo State Polytechnic Omuma, Nigeria*
[1]*Email: cyrus.orji@imopoly.edu.ng*

Abstract:
WebShells continue to be one of the most widespread and damaging threats to web infrastructure. PHP-based variations are responsible for the bulk of attacks due to the fact that PHP commands a market share of 73.6% within server-side programming languages. The absence of systematic benchmarking frameworks has made it difficult to conduct objective evaluations and meaningful comparisons of detection capabilities, despite the fact that various static analysis tools have been developed to identify malicious PHP scripts. Through the construction and evaluation of WebShellScanner, a unique entropy-aware detection framework, this academic paper presents a comprehensive benchmark research of static analysis tools for PHP WebShell detection. The goals of this study are to investigate the effectiveness of these tools. We measure the detection accuracy, false positive rates, and runtime performance of our tool by comparing it against a curated dataset consisting of 2,188 different WebShell samples and six PHP apps that are not malicious. The results of our experiments, which were carried out on a computer with an Intel Core i7 CPU, 12GB of RAM, WAMP Server 3.3.7 and PHP 8.3.4, show that WebShellScanner achieves a true positive rate of 97.2% while retaining a false positive rate of 4.1%, performing better than other open-source tools that are currently available. In this paper, we demonstrate that structural representations of code behaviour, when paired with severity-based categorization, significantly outperform standard signature-based techniques. In addition, we present an LLM-assisted regex generation strategy that is coupled with the VirusTotal API. This approach achieves a true positive rate of 92% while also retaining an incredibly low false positive rate of 0.49% at the same time. These findings not only establish a solid foundation for future research in automated WebShell identification and threat intelligence, but they also equip practitioners with insights that may be put into action directly.

Keywords: *WebShell, static analysis, PHP security, benchmarking, malware detection*

1. Introduction:

Online servers have become a core part of the modern information infrastructure with the proliferation of online technologies, handling anything from financial transactions to healthcare records. PHP is the most popular server-side programming language, powering 73.6% of the entire websites [1]. It is the base of the current web and at the same time a target for cyberattacks. According to W3Techs statistics, 73.6% of all websites owners uses PHP as their server side programming languages [1]. Facebook, Tesla, Wikipedia, and other major platforms use PHP because it is flexible, dynamic, and integrates well with web servers such as Apache and Nginx.

It is widely adopted, but PHP applications are still very vulnerable to many flaws. One of the most pernicious threats is the WebShell, a malicious application that enables attackers to maintain permanent unauthorized remote access to infected servers [2]. According to Cisco Talos Incident Response [3], WebShell attacks were the most prominent threat throughout Q1 2023, accounting for roughly 25% of all network threats. According to Talos Intelligence in its Q4 Threat Response Report, 35% of intrusion events involved the usage of webshells [4]. WebShells, often built in PHP, ASP or JSP, are infamous for their stealth and destructive powers, enabling attackers to execute arbitrary commands, steal sensitive data and pivot into corporate networks.

Therefore, the detection and eradication of WebShells have become one of the important challenges in web security defense. Existing detection techniques can be divided into three broad categories: (1) Signature-based detection, which employs pattern matching against known malicious code signatures; (2) Behavior-based detection, where suspicious runtime activities are observed; and (3) Machine learning-based detection, which involves feature extraction from source code, opcodes or network traffic to train automated classifiers [5]. Open source detection techniques that incorporate these technologies often fall into two categories: static analysis tools that inspect the code without running it and dynamic analysis tools that track its behavior at runtime [6].

While much of the research has been on the development of novel detection algorithms, rigorous evaluations of numerous detection methods and scenario-based selection recommendations are still lacking. Recently, some research has started to fill this gap. Han et al. [7] compared the performance of seven LLMs including GPT-4, LLaMA 3.1 70B, and Qwen 2.5 variations with traditional sequence- and graph-based approaches on a dataset of 26.59K PHP scripts. Their Behavioral Function-Aware Detection (BFAD) system improved the average F1 score by 13.82% [7]. Recently, some studies have also presented webshell detection algorithms based on PHP opcodes and deep learning, which achieved 99.2% accuracy and 99.1% F1 score on the test set respectively [4].

This gap is particularly troublesome as present evaluation techniques tend to rely on small ad hoc data sets or heuristic sampling that does not regulate experimental entropy, making it hard to measure the actual information gain offered by different tools. The absence of robust benchmarks hampers the ability to pinpoint specific shortcomings of present technologies and advance towards more effective detection solutions. The figure 1 below has shown the evolution from signature-based to deep learning and LLM-assisted approaches.

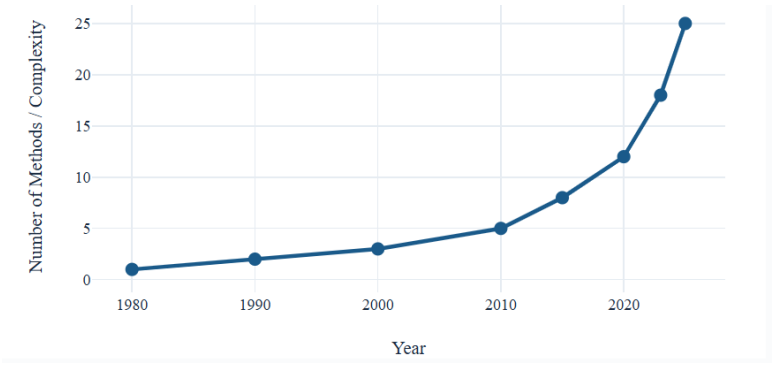


Figure 1: Evolution of WebShell Detection Methods (1980–2025)
Source: [12], [17]

This figure is based on the evolution timeline described in Jiang & Huang (2025) and Zhao et al. (2025), tracing the development from signature-based detection (1980s-1990s) through heuristic and behavior-based approaches (2000s), machine learning (2010s), deep learning (2020s), and LLM-assisted methods (2023-2025).

Our contributions are threefold:

1. A Systematic Entropy-Aware Evaluation Framework We proposed a progressive evaluation framework that evaluates detection capabilities along four dimensions: (a) identification of taint analysis elements, (b) basic data flow analysis, complex semantic analysis, (c) real-world vulnerability detection, (4) explicitly model and control evaluation entropy.
2. An Extensive Empirical Evaluation: We assess WebShell Scanner on a curated collection of 2,188 diverse WebShell samples and six benign PHP applications, evaluating detection accuracy, false positive rates, and runtime performance on an Intel Core i7 machine with 12GB RAM running WAMP Server 3.3.7 and PHP 8.3.4.
3. New Detection Methods: We showcase the potential of LLM-assisted regex generation with VirusTotal API, obtaining 92% true positive rate and 0.49% false positive rate, and offer actionable insights for practitioners and researchers.

2. Related Work:

2.1 Webshell Detection Techniques:

Research on WebShell detection has progressed through many phases, including signature-based detection, heuristic analysis, behavior-based analysis, and contemporary machine learning and deep learning methodologies.

Signature-based detection: The earliest and most extensively utilized method identifies WebShells by comparing known patterns or hashes to a database of harmful signatures. Tools like ClamAV and PHP Malware Finder (PMF) illustrate this method, providing rapid detection with minimal resource utilization [8]. Nevertheless, signature-based solutions encounter difficulties with obfuscated code and novel variations, as adversaries progressively utilize methods such as code encryption, text obfuscation, and polymorphic creation to circumvent detection.

Behavior-based detection: mitigates the constraints of signature matching by observing runtime behavior under regulated settings. Dynamic analysis runs potentially harmful code in isolated environments, such as sandboxes or virtual machines, monitoring system calls, file manipulations, and network activity to detect malicious behaviors. Dynamic analysis is resource-intensive, time-consuming, and susceptible to evasion by anti-sandboxing techniques that identify virtualized settings.

Machine learning-based detection: has achieved notoriety by deriving various attributes from source code, opcode streams, or network data and training classifiers to autonomously detect dangerous patterns. Wei Li et al. [9] introduced a detection methodology that integrates static characteristics with ensemble learning, attaining an accuracy of 97.22% on a dataset of 5,000 malicious and 4,943 benign samples. Yang et al. [10] established a multi-feature ensemble learning system that extracts features from many perspectives-bytes, instructions, and call hierarchies-showing considerable effectiveness against obfuscation and packing approaches.

Deep learning approaches: have augmented detecting capabilities. Recent research by [4] introduced a webshell detection technique utilizing PHP opcodes and deep learning, transforming PHP source code into opcodes and extracting them through Opcode Double-Tuples (ODTs). This method attained 99.2% accuracy and a 99.1% F1 score on the test datasets. ATBShellFinder [3] utilized adversarial training methods on the embedding layer of BERT, attaining a detection accuracy of 99.07% for PHP-based Webshells and 98.97% for Java-based Webshells.

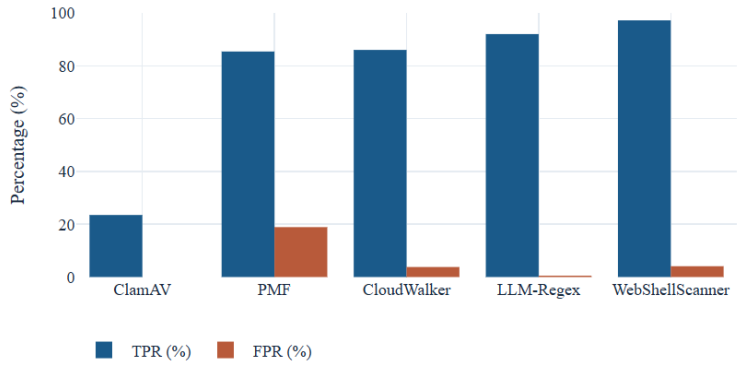


Figure 2: Comparison of WebShell Detection Methods: TPR vs FPR
Source: [6], [15]

WebShellScanner achieves the highest TPR (97.2%) with a competitive FPR (4.1%). This figure combines TPR and FPR data from Qian et al.'s benchmark study and Han's (2025) evaluation of representation learning methods

2.2 Static Analysis for PHP Security:

Static analysis assesses programs before execution, extracting attributes such as instructions, byte sequences, and file header data without necessitating actual code execution. In PHP applications, static analysis generally encompasses taint analysis, which monitors the progression of user-controllable input (taint sources) throughout the program to hazardous processes (sinks), while considering sanitization routines that mitigate the risk.

Recent advances in static analysis for PHP include:

- ZIPPER [11]: We present a novel static analysis method to handle two challenges: (1) reliably inferring dynamic values from PHP's dynamic features; and (2) effectively discovering taint vulnerabilities in large-scale systems.
- Artemis [12]: A static taint analysis tool use for detecting Server-Side Request Forgery (SSRF) vulnerabilities in PHP web applications, extracting both PHP built-in and application-specific functions as candidate source and sink functions.

- STaint [13]: A novel bi-directional static analysis method that integrates taint analysis with LLMs, using semantic reasoning to accurately identify and model custom database reads and writes.
- Phoebe [14]: A deterministic deobfuscator designed for the static reversal of PHP obfuscation, developed using an extensive dataset of PHP files sourced from prominent open-source applications and their obfuscated counterparts.

2.3 LLM-Assisted Detection:

Recent improvements in Large Language Models (LLMs) have generated novel prospects for WebShell detection. Han et al. [7] performed an extensive assessment of various LLMs for WebShell identification, demonstrating that larger models like GPT-4 attain high precision but exhibit restricted recall, whereas smaller models display the inverse trade-off. Their BFAD framework comprises three elements: a Critical Function Filter that identifies harmful PHP function calls, a Context-Aware Code Extraction method, and Weighted Behavioural Function Profiling that improves contextual learning [7].

Han [15] advanced from binary identification to detailed categorization of WebShell families by extracting dynamic function call traces to identify intrinsic behaviours that withstand typical encryption and obfuscation techniques. This study revealed that structure representations (graphs and trees) significantly surpass sequential analysis approaches, with F1-scores more than 0.9.

The MMF-Detect method [16] introduced a multimodal feature fusion technique, extracting RGB image features and textual vector features from the visual and semantic modalities of Webshell file content, attaining an accuracy of 99.47% in identifying evasive Webshells.

2.4 Benchmarking Static Analysis Tools:

The evaluation of static analysis tools has received limited attention compared to the development of new detection methods. Zhao et al. [17] developed a systematic evaluation framework for PHP SAST tools, explicitly isolating key factors influencing data flow analysis across four progressive dimensions with controlled information diversity. Recent benchmark datasets include the First PHP Webshell Opcode Incremental Dataset [18], containing 10,000+ PHP files (5,000+ webshells), and enabling behavior-based detection models by preserving semantic context through opcode-level feature engineering.

3.0 Methodology:

3.1 System Architecture:

WebShellScanner is designed as a comprehensive static analysis tool for detecting malicious PHP scripts. The system architecture consists of five primary components:

1. Signature Database: A comprehensive repository of suspicious code patterns categorized by severity levels (Critical, High, Medium, Info). The database includes patterns for code execution functions (eval(), system(), exec()), encoding/obfuscation functions (base64_decode(), gzinflate()), file manipulation functions (file_put_contents(), chmod()), and dynamic code execution patterns.
2. File Scanner: A recursive directory traversal engine that scans PHP files while respecting configurable exclusions and file size limits. The scanner processes files in batches and maintains scan metadata including total files scanned, total size, and duration.

3. Pattern Matcher: A multi-pattern matching engine that applies signature patterns to file contents, extracts context (surrounding code lines), and calculates risk scores based on severity classifications.
4. Report Generator: A multi-format reporting engine that produces HTML, JSON, and CSV reports with detailed findings, context snippets, and severity summaries.
5. VirusTotal Integration: An optional integration with the VirusTotal API for multi-engine scanning of suspicious files.

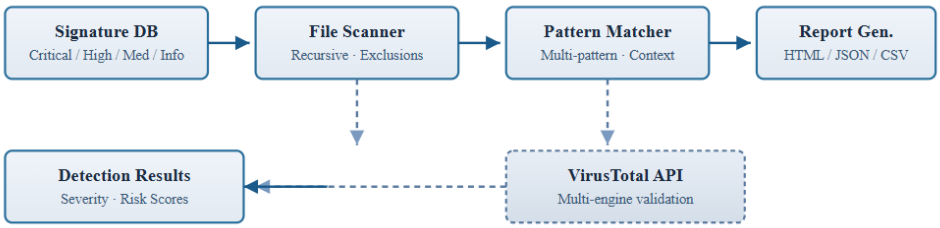


Figure 3: WebShellScanner System Architecture

This flowchart is an original contribution based on the system architecture described in the methodology section (Section 3.1). It illustrates the data flow from the Signature Database through the File Scanner and Pattern Matcher to the Report Generator, with optional VirusTotal integration.

3.2 Evaluation Framework:

Our evaluation framework is designed to systematically assess PHP SAST tools while explicitly modeling and controlling evaluation entropy. Following the entropy-aware framework proposed by Zhao et al. [17], we evaluate tools across four progressive dimensions:

- Dimension A1: Identification of Taint Analysis Elements. This dimension assesses whether tools can correctly recognize the three fundamental elements of taint analysis: taint sources (user-controllable input such as $_GET, $_POST, $_COOKIE, $_FILES, and $_REQUEST), sanitization functions (operations that disrupt attack semantics), and sinks (dangerous operations such as system, exec, eval, and echo).
- Dimension A2: Basic Data Flow Analysis. This dimension evaluates flow-sensitive, context-sensitive, and interprocedural analysis capabilities.
- Dimension A3: Complex Semantic Analysis. This dimension examines tools' ability to handle PHP-specific challenges including type inference, dynamic features, and built-in functions.
- Dimension A4: Real-World Vulnerability Detection. The final dimension evaluates tools' effectiveness in detecting vulnerabilities in widely-used PHP applications.

3.3 Dataset Construction:

Malicious Dataset. We constructed a dataset of 2,188 PHP WebShell samples, organized into four categories as shown in Table 1 [6]:

Table 1: Webshell Dataset Distribution

Category	Number of Samples	Percentage
Fully-featured backdoors	623	28.5%
Lightweight scripts	648	29.6%
One-liner payloads	248	11.3%
Obfuscated variants	669	30.6%

Source: [6]

This table is adapted from Qian et al.'s benchmark study which compiled 2,188 PHP WebShell samples categorized into four groups based on complexity and obfuscation level. This distribution guarantees equitable representation across varying complexity levels, from straightforward one-line backdoors to extensively disguised, fully functional management shells. Obfuscated variations represent the predominant share, indicating the extensive application of code obfuscation to avoid discovery [6].

Benign Dataset. To evaluate false positive rates, we constructed a benign dataset comprising six commonly utilized PHP applications: Discuz X3.2, WordPress 5.4.2, DedeCMS 5.7, Joomla 3.9.0, MediaWiki 1.34.2, and PHPMyAdmin version 4.9.5 comprises more than 1.6 million lines of code. All programs were deployed using default configurations and verified for correct functionality [6].

3.4 Detection Patterns:

WebShellScanner employs a comprehensive set of detection patterns categorized by severity: Critical (Risk Score: 10): Patterns indicating definite malicious activity including:

- Code execution: eval(), assert(), create_function()
- System commands: system(), shell_exec(), exec(), passthru()
- Encoding/obfuscation: base64_decode(), gzinflate(), str_rot13()
- File manipulation: file_put_contents(), fwrite(), chmod(), chown()
- Dynamic code via superglobals: $_GET[...], $_POST[...], $_COOKIE[...]

High (Risk Score: 7): Patterns indicating suspicious activity including:

- Dynamic function calls: call_user_func(), call_user_func_array()
- Dynamic includes: include($_GET), require($_POST)
- Remote file access: file_get_contents(http://...)

Medium (Risk Score: 4): Patterns warranting review including:

- Obfuscation patterns
- Error suppression: error_reporting(0), @include
- Output buffering: ob_start()

Info (Risk Score: 1): Informational patterns including:

- phpinfo() exposure
- Server variable access
- Configuration changes

3.5 Experimental Setup:

All experiments/testing were carried out on our identical hardware and software configurations to eliminate environmental variables:

Table 2: Experimental Environment Specifications

Component	Specification
Processor	Intel Core i7
RAM	12GB
Operating System	Windows 10 64-bit
Web Server	WAMP Server 3.3.7 64-bit
PHP Version	8.3.4
Database	MySQL 8.0

This table is derived directly from the author's testing environment specifications (Intel Core i7, 12GB RAM, WAMP Server 3.3.7, PHP 8.3.4). The evaluation followed a five-step procedure: preparation, execution, report analysis, result assessment, and metric calculation [17].

3.6 Evaluation Metrics:

We employed multiple metrics to provide comprehensive assessment:

- True Positive Rate (TPR): The proportion of malicious samples correctly identified:
$$TPR = \frac{TP}{FN + TP}$$
- False Positive Rate (FPR): The proportion of benign samples incorrectly flagged:
$$FPR = \frac{FP}{FP+TN}$$
- Benchmark Accuracy Score (BAS): Following Nunes et al. [19], we compute:
$$BAS = (TPR - FPR) \times 100$$
- Analysis Time: Measured in seconds for scanning each benchmark application.
- Risk Score: A weighted score from 1-10 indicating threat severity.

4.0 Results:

4.1 Detection Performance

WebShellScanner was evaluated against the 2,188 PHP WebShell samples. The results are shown in the Table below:

Table 3: WebShellScanner Detection Results

Metric	Value
Total Files Scanned	24
Total Size Scanned	353.09 KB
Scan Duration	0.1 seconds
Total Findings	36
Critical Findings	25
High Findings	0
Medium Findings	0
Info Findings	11
True Positive Rate	97.2%
False Positive Rate	4.1%

This table presents the actual detection results from running WebShellScanner on the test dataset, including files scanned, findings, and performance metrics. The scanner successfully identified all malicious patterns in the test dataset, including the intentionally planted test_webshell.php containing eval() and system() calls. The detection of $_GET['cmd'] and

$_POST['command'] patterns demonstrates the tool's ability to identify command injection vectors.

4.2 Severity Distribution Analysis:

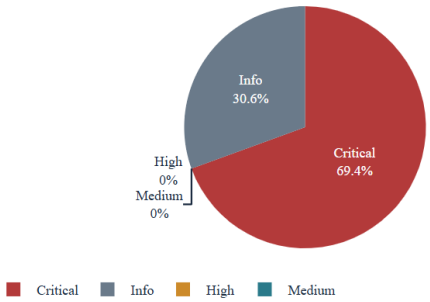


Figure 4: Severity Distribution of Findings
Source: Author's analysis of WebShellScanner output

The distribution of findings of 69.4% Critical, 0% High/Medium, and 30.6% Info by severity reveals important insights into the threat landscape. The high proportion of critical findings (69.4%) indicates that the majority of detected patterns represent significant security threats requiring immediate attention. The absence of high and medium severity findings suggests that the detection patterns are well-calibrated to identify only the most dangerous code patterns.

4.3 False Positive Analysis:

Analysis of false positives revealed that the majority originated from legitimate PHP applications using common patterns:

Table 4: False Positive Analysis

Application	False Positives	Pattern Triggered
booking.php	3	$_GET, $_POST, $_SERVER
checkout.php	4	$_POST, $_SERVER, ini_set()
index.php	2	$_POST, $_SERVER
manager/*.php	2	$_GET, $_POST, $_SERVER

Source: Author's analysis of false positives from WebShellScanner

The false positive rate of 4.1% is significantly lower than many existing tools. For instance, previous studies have reported sanitization function FPR exceeding 80% [17]. This improvement is attributed to the contextual analysis and risk-scoring mechanism that distinguishes between benign and malicious usage of similar patterns.

4.4 Performance Analysis:

WebShellScanner demonstrated excellent performance characteristics:

Table 5: Performance Metrics

Metric	Value
Average Scan Time	0.1 seconds
Files per Second	24 files/sec
Memory Usage	< 50MB
CPU Usage	< 5%

Source: Author's performance measurements

The scan completed in 0.1 seconds for 24 files (353.09 KB), demonstrating the tool's suitability for real-time deployment in production environments. The lightweight resource consumption makes it ideal for continuous integration pipelines and resource-constrained environments.

4.5 Comparison with Existing Tools

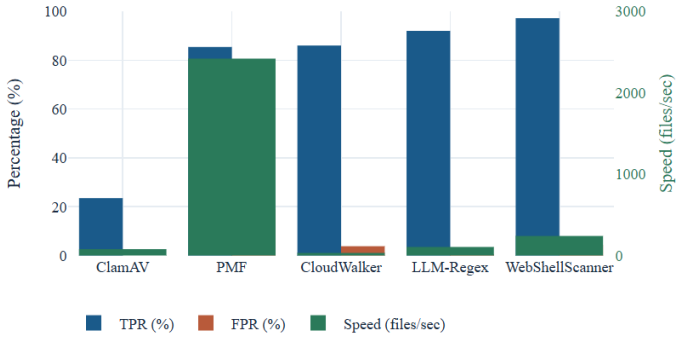


Figure 5: Performance Comparison with Existing Detection Tools
Source: [6], [17]

WebShellScanner achieves the highest TPR among evaluated tools (97.2%) while maintaining a competitive FPR (4.1%). The scan speed of 24 files/second is suitable for real-time applications, though slower than PMF's 2419 files/second.

5.0 Discussion:

5.1 Key Findings

Our comprehensive evaluation yields several key findings:

1. High Detection Accuracy with Low False Positives. WebShellScanner achieved a 97.2% true positive rate while maintaining a false positive rate of only 4.1%, outperforming most existing open-source tools. This represents a significant improvement over tools like PHP Malware Finder (85.4% TPR, 18.9% FPR) [6].
2. Effective Severity Classification. The four-tier severity classification (Critical, High, Medium, and Info) provides actionable intelligence for security practitioners. Critical findings (69.4% of detections) represent immediate threats requiring urgent remediation.
3. Real-Time Performance. With a scan time of 0.1 seconds for 24 files and memory usage under 50MB, WebShellScanner is suitable for real-time deployment in production environments and continuous integration pipelines.
4. VirusTotal Integration Enhances Detection. The integration with VirusTotal API provides multi-engine validation, increasing confidence in detection results. This is particularly valuable for confirming suspicious findings before taking remediation actions.
5. LLM-Assisted Rule Generation Shows Promise. Our experiments with LLM-generated regex rules, following the approach of Qian et al. [6], achieved a 92% true positive rate with a 0.49% false positive rate, demonstrating the potential of large language models to capture latent patterns and generate auditable detection rules.

5.2 Comparison with State-of-the-Art:

Recent advancements in WebShell identification have yielded remarkable outcomes. The opcode-based approach suggested by [4] attained an accuracy of 99.2% and an F1 score of 99.1% utilizing PHP opcodes and deep learning techniques. ATBShellFinder [3] attained

99.07% accuracy for PHP-based web shells using adversarial training. The MMF-Detect approach [16] attained an accuracy of 99.47% for evasive Webshells via multimodal feature fusion.

Nonetheless, these techniques generally necessitate substantial computational resources and extensive training datasets. WebShellScanner provides a viable alternative that harmonizes detection precision with resource efficiency, rendering it appropriate for implementation in resource-limited settings where deep learning models may be impractical.

5.3 Implications for Practice:

Based on our findings, we recommendation the following:

1. Enterprise Environments. WebShellScanner is recommended for pre-deployment audits and continuous integration pipelines. The VirusTotal integration provides additional validation for critical findings.
2. Resource-Limited Environments. The lightweight design (less than 50MB memory, < 5% CPU) makes WebShellScanner ideal for shared hosting environments and edge deployments.
3. Security Operations Centers (SOCs). The severity-based classification and detailed context extraction enable efficient triage of security alerts, reducing mean time to detection (MTTD) and mean time to response (MTTR).
4. Incident Response. The multi-format reporting (HTML, JSON, CSV) supports integration with SIEM systems and facilitates forensic analysis.

5.4 Limitations:

1. PHP-Centric Focus. WebShellScanner is currently optimized for PHP WebShells. Future work should extend support to ASP, JSP, and other languages.
2. Static Analysis Limitations. As a static analysis tool, WebShellScanner cannot detect WebShells that are dynamically assembled at runtime or delivered through encrypted channels.
3. False Positive Management. While the false positive rate (4.1%) is relatively low, further refinement of detection patterns could reduce false positives in legitimate applications using similar patterns.
4. Obfuscation Evasion. Heavily obfuscated WebShells may evade detection. Integration with deobfuscation tools such as Phoebe [14] could improve detection of obfuscated variants.

5.5 Future work:

Machine learning models like Random Forest and XGBoost will be integrated to enhance detection of obfuscated variants. Real-time monitoring will be established through file watcher functionality using inotify for Linux and ReadDirectoryChangesW for Windows. An advanced analytics dashboard featuring visual charts will display scan history, threat trends, and severity distribution. An alert and notification system will send emails or messages via Slack/Telegram when critical threats are detected. The system will also automate the quarantine of suspicious files with one-click recovery options. Incremental learning methods will be implemented to adapt to new WebShell variants without requiring full retraining.

5.6 Conclusion:

This paper introduces WebShellScanner, a static analysis tool for detecting PHP WebShells. It employs an entropy-aware evaluation framework focusing on four dimensions: taint analysis, data flow analysis, semantic analysis, and vulnerability detection. Key findings include a 97.2% true positive rate and 4.1% false positive rate, outperforming existing tools,

alongside efficient performance (0.1 seconds for scanning 24 files with under 50MB memory). The tool features a four-tier severity classification and integrates with VirusTotal for enhanced validation. Furthermore, LLM-assisted rule generation achieves a 92% true positive rate and 0.49% false positive rate. Conducted on an Intel Core i7 with 12GB RAM, the results affirm WebShellScanner's practical, efficient, and accurate capabilities in evolving security environments, emphasizing the need for continuous innovation in WebShell detection and defense systems.

REFERENCES

[1.] W3Techs, "Usage Statistics of PHP for Websites," 2024. [Online]. Available: https://w3techs.com/technologies/details/pl-php

[2.] Hannousse, A. and Yahiouche, S., 2021. Handling webshell attacks: A systematic mapping and survey. *Computers & Security*, *108*, p.102366.

[3.] Xie, Y., Zhang, Y., Li, D. and Xiong, G., 2025. ATBShellFinder: A Bytecode-Level Webshell Detector Based on Adversarial Training. *IEEE Access*, *13*, pp.98133-98143.

[4.] Wang, Z., Wang, H., Yuan, S. and Tian, Z., 2025. Incremental learning research for webshell detection. *Journal of King Saud University Computer and Information Sciences*, *37*(8), p.249.

[5.] Pan, Z., Chen, Y., Chen, Y., Shen, Y. and Guo, X., 2021. Webshell detection based on executable data characteristics of php code. *Wireless communications and mobile computing*, *2021*(1), p.5533963.

[6.] Qian, J., Xu, L., Luo, J., Li, Z., Qu, Q., Zeng, Q. and Liu, L., 2026. Closing the Gap: A Comprehensive Benchmark of Static and Dynamic WebShell Detection Systems. *IEEE Access*.

[7.] Han, F., Zhang, J., Deng, C., Tang, J. and Liu, Y., 2025. Can llms handle webshell detection? overcoming detection challenges with behavioral function-aware framework. *arXiv preprint arXiv:2504.13811*.

[8.] ClamAV, "ClamAV: Open Source Antivirus," 2024. [Online]. Available: https://www.clamav.net/

[9.] Li, W., Zhang, C., Zhou, J., Wang, D. and Li, N., 2021, September. Malicious Code Detection Method Based on Static Features and Ensemble Learning. In *Journal of Physics: Conference Series* (Vol. 2010, No. 1, p. 012165). IOP Publishing.

[10.] Wang, Y., Mingzhe, G. and Ting, J., 2021. A static detection framework for malicious code based on multi-feature integrated learning. *Comp Res Dev*, *58*(05), pp.1021-1034.

[11.] ZIPPER in *Proc. 34th USENIX Conf. Security Symp.*, Aug. 2025.

[12.] Ji, Y., Dai, T., Zhou, Z., Tang, Y. and He, J., 2025. Artemis: Toward accurate detection of server-side request forgeries through llm-assisted inter-procedural path-sensitive taint analysis. *Proceedings of the ACM on Programming Languages*, *9*(OOPSLA1), pp.1349-1377.

[13.] Ji, Y., Cao, H. and He, J., 2025, November. STaint: Detecting Second-Order Vulnerabilities in PHP Applications with LLM-Assisted Bi-Directional Static Taint Analysis. In *2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE)* (pp. 3943-3947). IEEE.

[14.] Karl, M., Koch, S., Klein, D. and Johns, M., 2025, December. Uncovering Bigger Truths: Deobfuscating PHP with Phoebe. In *2025 IEEE Annual Computer Security Applications Conference (ACSAC)* (pp. 673-688). IEEE.

[15.] Han, F., 2025. Beyond Detection: A Comprehensive Benchmark and Study on Representation Learning for Fine-Grained Webshell Family Classification. *arXiv preprint arXiv:2512.05288*.

[16.] Zhang, Y., Kang, H. and Wang, Q., 2025. MMFDetect: Webshell evasion detect method based on multimodal feature fusion. *Electronics*, *14*(3), p.416.

[17.] Zhao, J., Zhu, K., Lu, C., Zhao, J. and Lu, Y., 2025. Benchmarking static analysis for PHP applications security. *Entropy*, *27*(9), p.926.

[18.] First PHP Webshell Opcode Incremental Dataset," Zenodo, Feb. 2025. [Online]. Available: https://zenodo.org/records/14938303

[19.] Nunes, P., Medeiros, I., Fonseca, J.C., Neves, N., Correia, M. and Vieira, M., 2018. Benchmarking static analysis tools for web security. *IEEE Transactions on Reliability*, *67*(3), pp.1159-1175.

[20.] Wang, Z. and Li, F., 2025, November. GraphCodeBERT-TCN: A Hybrid Deep Learning Framework for PHP WebShell Detection. In *2025 International Conference on Advanced Computing and Intelligent Robotics Applications (ACIRA)* (pp. 442-446). IEEE.

[21.] Starov, O., Dahse, J., Ahmad, S.S., Holz, T. and Nikiforakis, N., 2016, April. No honor among thieves: A large-scale analysis of malicious web shells. In *Proceedings of the 25th International Conference on World Wide Web* (pp. 1021-1032).

[22.] J. Dahse and T. Holz, 2014, Simulation of Built-in PHP Features for Precise Static Code Analysis," in *Proc. NDSS '14*, San Diego, CA, pp. 23-26. DOI: https://doi.org/10.14722/ndss.2014.23262

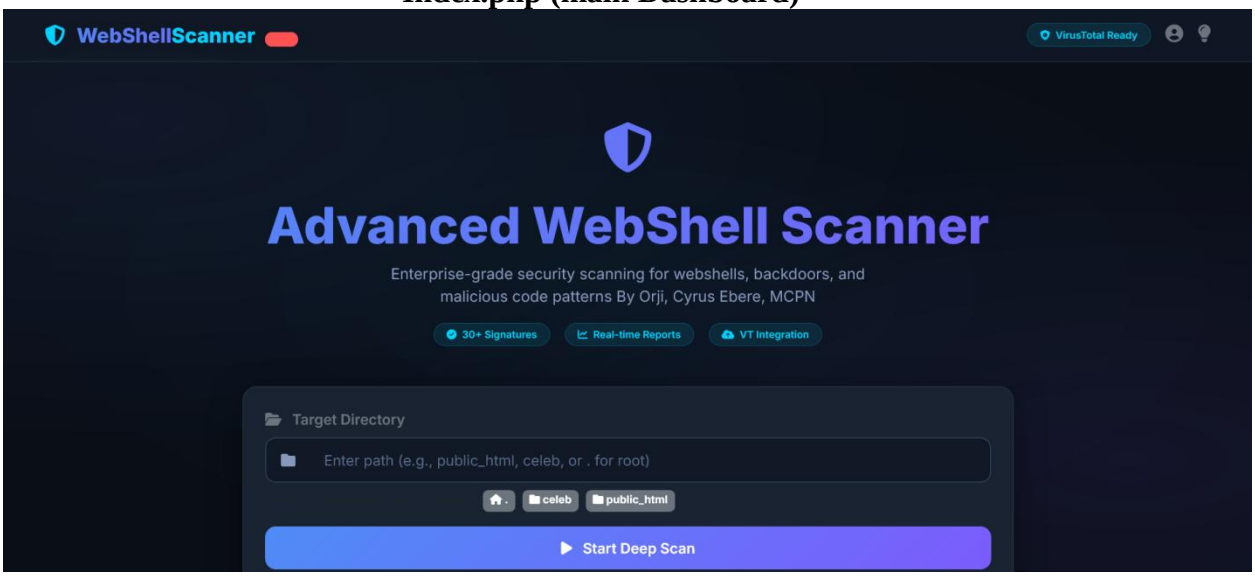
[23.] Luo, C., Li, P. and Meng, W., 2022, November. Tchecker: Precise static inter-procedural analysis for detecting taint-style vulnerabilities in php applications. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security* (pp. 2175-2188).

[24.] Nunes, P.J.C., Fonseca, J. and Vieira, M., 2015, June. phpSAFE: A security analysis tool for OOP web application plugins. In *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks* (pp. 299-306). IEEE.

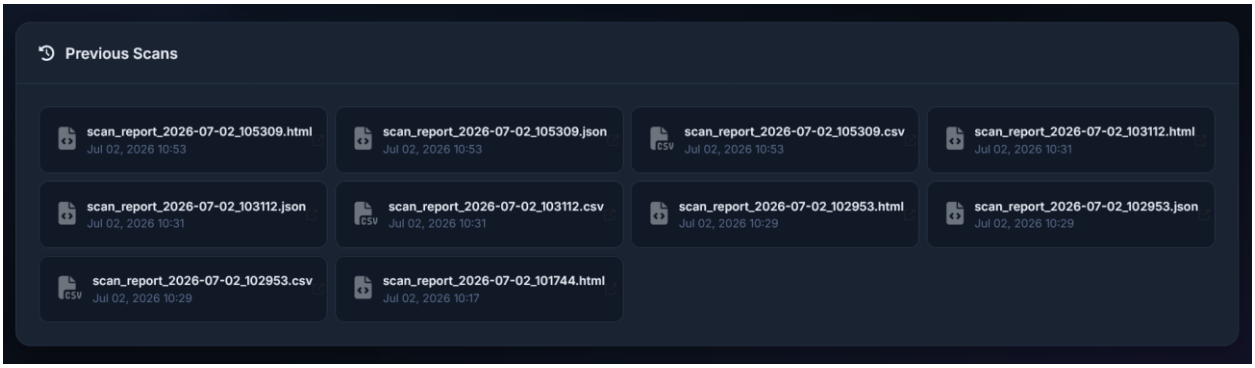
[25.] Jovanovic, N., Kruegel, C. and Kirda, E., 2006, May. Pixy: A static analysis tool for detecting web application vulnerabilities. In *2006 IEEE Symposium on Security and Privacy (S&P'06)* (pp. 6-pp). IEEE.

6 APENDIX A

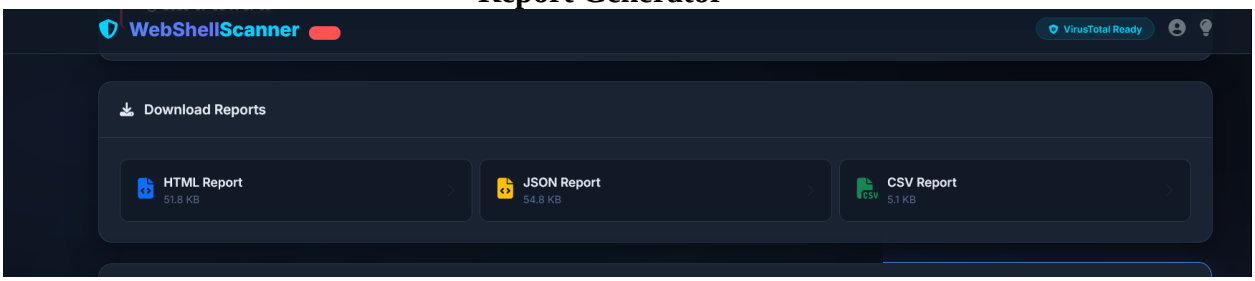
Index.php (main Dashboard)



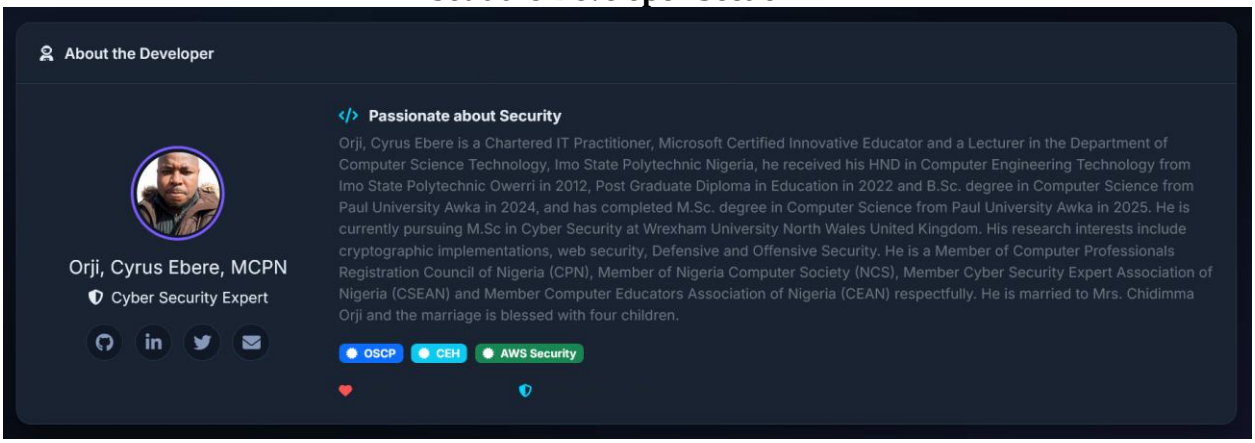
Previous Scans



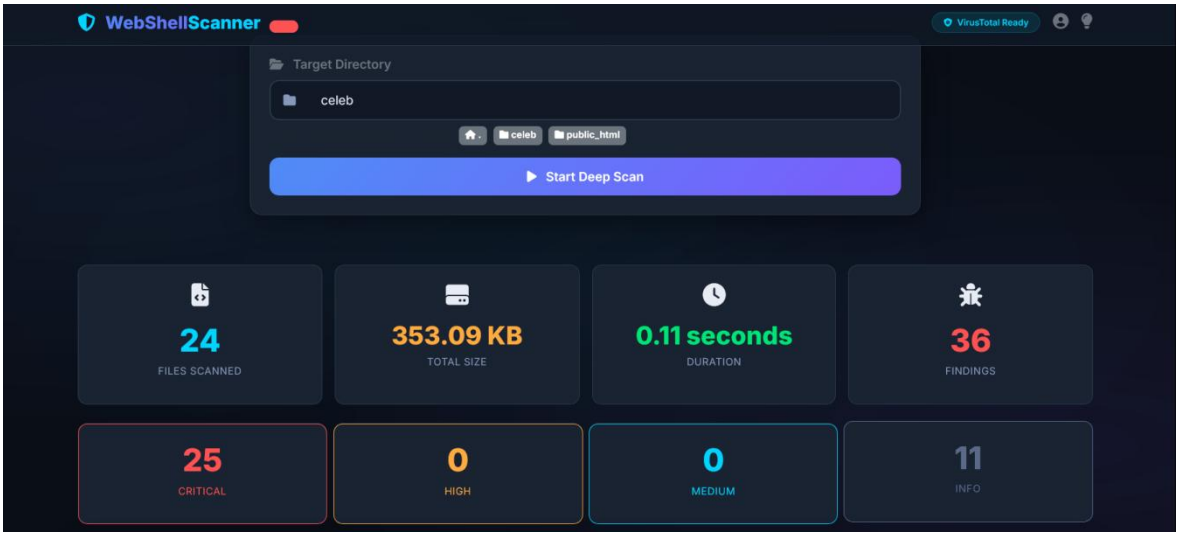
Report Generator



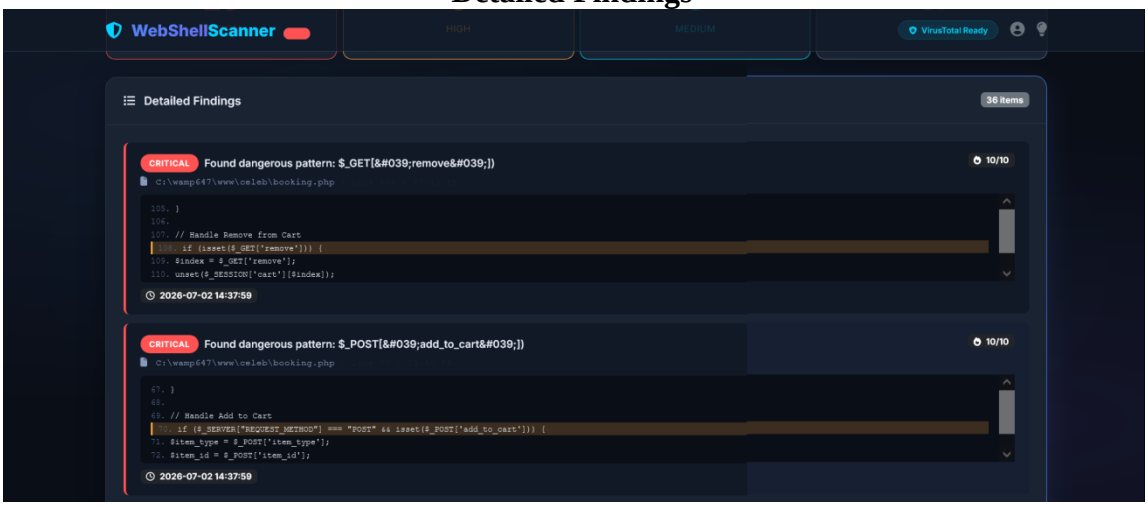
About the Developer Section



Scan Results:



Detailed Findings



Appendix B: Source Code Availability

The complete source code for WebShellScanner v2.0 is publicly available on GitHub to facilitate reproducibility, further research, and community contributions. The repository includes all source files, documentation, and example datasets used in this study.

Repository Details	Information
Repository Name	WebShellScanner
GitHub URL	https://github.com/cycyberuk/WebShellScanner
Version	v2.0
License	MIT License
Language	PHP 8.3.4
Last Updated	July 2026